Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

Bi-objective Workflow Scheduling in Production Clouds: Early Simulation Results and Outlook

Juan J. Durillo, Radu Prodan

Institute of Computer Science, University of Innsbruck, Austria
juan,radu@dps.uibk.ac.at

Sorina Camarasu-Pop, Tristan Glattard

CREATIS – CNRS, Lyon-Vileurbanne, France
tristan.glatard,sorina.pop@creatis.insa-lyon.fr

Frédéric Suter

IN2P3 Computing Center, Lyon-Villeurbanne, France
frederic.suter@cc.in2p3.fr

Abstract

We present MOHEFT, a multi-objective list scheduling heuristic that provides the user with a set of Pareto tradeoff optimal solutions from which the one that better suits the user requirements can be manually selected. We demonstrate the potential of our method for multi-objective workflow scheduling on the commercial Amazon EC2 Cloud by comparing the quality of the MOHEFT tradeoff solutions with a state-of-the-art multi-objective approach called SPEA2 for three types of synthetic workflows with different parallelism and load balancing characteristics. We conclude with an outlook into future research towards closing the gap between the scientific simulation and real-world experimentation.*

Keywords Scheduling, scientific workflows, Cloud computing, multi-objective optimisation, list-based heuristics

I. Introduction

In previous e-science research [21], many scientists have found in scientific workflows an attractive model of building large scale applications for heterogeneous wide-area parallel and distributed computing systems such as Grids. Typically, a scientific workflow application [20] consists of several (legacy) programs (referred from now on as tasks or activities) in the form of a dependency graph, where the input of some of these programs may depend on the output of the others. Once the application is composed as a workflow, its performance depends on how the individual tasks are mapped (scheduled) on to the available parallel and distributed resources. Traditionally, finding an optimal schedule of the tasks minimising the *makespan* or completion time of the whole workflow has been the main objective and a major NP-complete challenge [23]. As a consequence, many heuristics and meta-heuristics [4] for approximating a solution to this problem have been proposed [18, 22].

In the context of Cloud computing, the computed mapping must additionally optimise the *economic cost* incurred by renting resources. Today, most commercial Clouds offer heterogeneous types of resources at different prices and with different performance. For example, in Amazon EC2 (`http://aws.amazon.com/ec2/pricing/`) a user can choose among different types of instances, where the fastest resource is about eight times more expensive than the slowest one[1]. In these circumstances, the workflow scheduling problem has to be formulated as a *multi-objective optimisation problem (MOP)* which aims at optimising at least two conflicting criteria: makespan and economic cost of workflow's execution. The main characteristic of MOPs is that no single solution exists that is optimal with respect to all objectives, but a set of tradeoff solutions known as *Pareto front* [9]. Solutions within this set cannot be further improved in any of the considered objectives without causing the degradation of at least another objective. Most related work [13, 17, 2] simplifies workflow scheduling optimising several competing objectives to a single-objective problem by aggregating all the objectives in one analytical function. The main drawback of these approaches is that the aggregation of the different objectives is made a priori, with any knowledge about the workflow, infrastructure, and in general about the problem being solved. Therefore, the computed solution may not properly capture the user preferences. On the other hand, few approaches computing the tradeoff solutions have been proposed. Their main advantage over the aggregative ones is that the user is provided with a set of optimal solutions from which the one that better suits the requirement or preferences can be manually selected.

To address this gap, we introduce in this paper a new multi-objective workflow scheduling method called *Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)* as an extension to the

[1] These prices only refer to the Standard On-Demand Instances (September 2013)

well-known HEFT [22] mono-objective workflow scheduling algorithm. Our proposal is a new heuristic-based method that computes a set of tradeoff solutions with a small additional overhead compared to the traditional single objective methods. In doing this, MOHEFT builds several intermediate workflow schedules in parallel in each step instead of a single one. To ensure the quality of the tradeoff solutions, MOHEFT uses dominance relationships and a metric called crowding distance to guarantee their diversity. MOHEFT is generic in the number and type of objectives, applied in this paper for optimising the makespan and economic cost of running workflow applications in an Amazon-based commercial Cloud.

While in theory a Cloud user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of N instances that can be simultaneously acquired. For example, in Amazon this maximum number is limited to $N = 20$ and could be enlarged through offline communication. Within this maximum number N, the user flexibly can choose between the different types of instances offered by the Cloud provider (e.g. `m1.small`, `m1.large`, `m1.xlarge`, `c1.medium`, `c1.xlarge` for Amazon EC2) with different performance and prices. The question which instances should compose the set of maximum size N for running the workflow becomes critical and has no single answer, since different combinations produce different tradeoff schedules. Moreover, the set of N instances does not need to be invariant during the whole workflow execution. For example, it may occur that one type of instance is particularly good at the beginning of the workflow execution, and a different type of instance is mostly beneficial towards the end. These additional constraints imposed by commercial Cloud systems require modifications to the proposed algorithms originally designed for heterogeneous distributed computing systems. Additionally, we also aim at highlighting the potential of the Pareto front as a tool for decision support, analysing how the user can exploit this information for improving the workflow schedule.

The paper is organised as follows. Section II defines the abstract workflow, resource, and problem definition underneath our approach. In Section III, we present our multi-objective workflow scheduling algorithm, adapted to the case of commercial Clouds. We present in Section IV the experimental setup for evaluating our technique on several synthetic and real-world workflows on Amazon EC2 (Section V). Finally, we summarise the conclusions and the future work in Section VI.

II. Model

This section formally describes the workflow, resource, and problem definition underneath our approach.

II.1 Workflow Model

We model a *workflow application* as a directed acyclic graph: $W = (A, D)$ consisting of n tasks (also referred in the remaining of this paper as activities) $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through control flow and data flow dependencies; $D = \{(A_i, A_j, Data_{ij}) \mid (A_i, A_j) \in A \times A\}$, where $Data_{ij}$ represents the size of the data which needs to be transferred from activity A_i to activity A_j. We use $pred(A_i) = \{A_k \mid (A_k, A_i, Data_{ki}) \in D\}$ to denote the *predecessor* set of activity A_i, (i.e. activities to be completed before starting A_i). Finally, we assume that the computational workload of every activity A_i is known and is given by the number of machine instructions required to be executed.

II.2 Resource Model

We assume that our hardware platform consists of a set of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$, which can be of any type as provided by Amazon EC2 (e.g. `m1.small`, `m1.medium`, `m1.large`, `m1.xlarge`, `c1.medium`, `c1.xlarge`). For a given resource R_j of a certain type, we know its average performance measured in GFLOPs. In our workflow model we assume that an activity that is executed in any of these resource can benefit from a parallel execution using all the virtual cores exposed by the instance, achieving the performance indicated in the last column of Table 1. The use of any of these resources is charged per every hour of computation following the Amazon prices indicated in the third column of that table. The final price is based not only on the resources' usage, but also in the data stored and transferred among different instances which depends on four components: (1) price per hours of resource's usage PE_{R_i}; (2) price per MB of data storage PS_{R_i}; (3) price per MB of data received PI_{R_i}; (4) price per MB of data sent PO_{R_i}.

The prices of these components depend on the Cloud provider. Currently, Amazon EC2 does not charge for internal data transfers among EC2 instances which do not require a public IP address, i.e. which do not require to be publicly reachable from Internet. Amazon EC2 also does not charge for incoming data from Internet to EC2 instances. In the case of outgoing data, the first GB transferred each month is free, and up to 10TB of information can be transferred at a relative low price of 0.120$ per GB. For the data storage, the price charged by Amazon EC2 is 0.10$ per stored GB.

Finally, commercial Clouds such as Amazon EC2 introduce constraints that must considered. While in theory a user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of N instances that can be simultaneously acquired. For example, in case of Amazon this maximum number is limited to 20 and can be enlarged through offline communication. Within this maximum number N, the user can flexibly choose between the different types of instances with different performance and prices. The question which instances to compose the set of maximum size N for running the workflow becomes critical and has no single answer since different systems of maximum size N will produce different tradeoff schedules. Moreover, the set of N instances does not have to be invariant during the whole workflow execution. For example, it may occur that one type of instance is particularly good at the beginning of the workflow execution, and a different type of instance the most beneficial at the end.

II.3 Problem Definition

Our problem consists in scheduling the execution of the workflow tasks on Cloud resources such that the makespan and the economic costs are minimised. In the rest of this paper, we will use $sched(A_i)$ to denote the resource on which the task A_i is scheduled to be executed. We describe in the following how the two objectives of interest are computed.

II.3.1 Makespan

For computing the workflow makespan, it is first necessary to define the *execution time* $t_{(A_i,R_j)}$ of an activity A_i on a resource $R_j = sched(A_i)$ as the sum of the time required for transferring the biggest input data from any $A_p \in pred(A_p)$ and the time required to complete A_i in R_j:

$$t_{(A_i,R_j)} = \max_{A_p \in pred(A_i)} \left\{ \frac{Data_{pi}}{b_{pj}} \right\} + \frac{workload(A_i)}{s_j}, \quad (1)$$

where $Data_{pi}$ is the size of the data to be transferred between A_p and A_i, b_{pj} is the bandwidth of one TCP stream between the resource where task A_p was executed and the resource R_j, $workload(A_i)$ the length of the task A_i in machine instructions, and s_j the speed of the resource R_j in number of machine instructions per second. Next, we can compute the *completion time* T_{A_i} of activity A_i considering the execution time of itself and its predecessors:

$$T_{A_i} = \begin{cases} t_{(A_i,sched(A_i))}, & pred(A_i) = \emptyset; \\ \max\limits_{A_p \in pred(A_i)} \left\{ T_{A_p} + t_{(A_i,sched(A_i))} \right\}, & pred(A_i) \neq \emptyset. \end{cases} \quad (2)$$

The workflow makespan is finally defined as the maximum completion time of all the activities in the workflow:

$$T_W = \max_{i \in [1,n]} \left\{ T_{(A_i,sched(A_i))} \right\}. \quad (3)$$

II.3.2 Economic Cost

The economic cost depends on two terms: the computation cost $C^{(comp)}$ and the cost of data transfer and storage $C^{(data)}$. We define $C^{(data)}_{(A_i,R_j)}$ as the cost of the data transfers $In(A_i)$ and $Out(A_i)$ and storage $Data(A_i)$ from executing activity A_i on resource R_j:

$$C^{(data)}_{(A_i,R_j)} = Data(A_i) \cdot t_{(A_i,R_j)} \cdot PS_{R_i} + In(A_i) \cdot PI_{R_i} + Out(A_i) \cdot PO_{R_i}, \quad (4)$$

For defining the cost $C^{(comp)}_{R_j}$ of using a resource R_j, we assume that for each task A_i executed on R_j we record two timestamps: $t^{(start)}_{A_i}$ when the activity starts and $t^{(end)}_{A_i}$ when the activity finishes its execution. The value $t^{(end)}_{A_i}$ can be computed as $t^{(start)}_{A_i} + t_{(A_i,R_j)} + \max_{A_i \in pred(A_p)} \left\{ \frac{Data_{ip}}{b_{jp}} \right\}$. We consider that the times for transferring the input $In(A_i)$ and the output data $Out(A_i)$ are included in the interval between $t^{(start)}_{A_i}$ and $t^{(end)}_{A_i}$. In other words, these time stamps indicate the period of time on which the resource R_j needs to be active due to the execution of the activity A_i.

Let us consider the set of p activities scheduled on resource R_j denoted as $\{J_1, \ldots, J_p\}$, where $p < n$ and $sched(J_i) = R_j, i \in [1,p]$, sorted based on their start timestamp: $t^{(start)}_{J_1} < \ldots < t^{(start)}_{J_p}$. Based on this ordering, we cluster them in $q \leq p$ different *groups* $G^{(j)}_k$, $1 \leq k \leq q$, so that all activities in one group are executed consecutively without releasing the resource. After the activity with the largest start timestamp in the group completes, the resource is released.

We construct the first group $G^{(j)}_1 = \{J_1, \ldots, J_r\}, r \leq p$, based on the following three rules:

1. The first activity J_1 belongs to the first group: $J_1 \in G^{(j)}_1$;
2. Every activity $J_i \in G^{(j)}_1, 2 \leq i \leq r$ completes before the resource is released. This means that J_i starts when the resource is still leased because of the execution of J_{i-1}:

$$t^{(start)}_{J_i} < t^{(start)}_{J_1} + \left\lceil \frac{t^{(end)}_{J_{i-1}} - t^{(start)}_{J_1}}{3600} \right\rceil \cdot 3600. \quad (5)$$

   We divide the total time in seconds of using a resource by 3600 in order to convert it to hours, and use the ceiling operator to round this value to complete hours of computation. Obviously, the resource will be rented for as many hours as required for finishing all the activities within this group.

3. The next activity (not part of the previous group) $J_{r+1} \notin G^{(j)}_1, r+1 \leq p$ starts in an instant of time $t^{start}_{J_{r+1}}$ when the resource has been already released, i.e., task J_r has finished its execution, the last rented period of one hour for executing J_r has expired, and the resource R_j was not needed in the period of time elapsed between $t^{end}_{J_r}$ and $t^{start}_{J_{r+1}}$. Mathematically, it can be expressed as:

$$t^{(start)}_{J_1} + \left\lceil \frac{t^{(end)}_{J_r} - t^{(start)}_{J_1}}{3600} \right\rceil \cdot 3600 < t^{(start)}_{J_{r+1}}. \quad (6)$$

Successive groups are built until the last activity J_p has been assigned. The second group $G^{(j)}_2$ is constructed in the same way starting from the task J_{r+1} instead of J_1. The same strategy is used for the rest of the groups. Once all the groups have been created, we define the cost $C^{(comp)}_{R_j}$ of using the resource R_j as the number hours required for executing all groups multiplied by the cost per hour:

$$C^{(comp)}_{R_j} = PE_{R_j} \cdot \sum_{k=1}^{q} \left\lceil \frac{\sum_{A_i \in G^{(k)}_{R_j}} t_{(A_i,R_j)}}{3600} \right\rceil. \quad (7)$$

We compute the economic cost of executing the entire workflow $W = (A, D)$ as the computation cost on all m resources plus the cost for transferring and storing the data:

$$C_W = \sum_{j=1}^{m} C^{(comp)}_{R_j} + \sum_{(A_i,A_j,Data_{ij}) \in D} C^{(data)}_{(A_i,R_j)}. \quad (8)$$

III. Cloud-aware Multi-Objective Heterogeneous Earliest Finish Time Algorithm

The original HEFT algorithm builds a solution by iteratively mapping the workflow tasks onto the available resources. That mapping is aimed at minimising the completion time of every task, so in every iteration only the resource which minimises this goal is considered. When multiple objectives are considered, the goal is to compute a set of tradeoff solutions by allowing the creation of several solutions at the same time instead of building a single one. Therefore, instead of mapping every task onto the resource where it is finishes earlier,

Algorithm 1 Cloud-aware MOHEFT algorithm.

```
Require: W = (A, D), A = ⋃_{i=1}^{n} A_i                          ▷ Workflow application
Require: N                                                        ▷ Maximum simultaneous instances
Require: I                                                        ▷ Number of different instance types
Require: R = ⋃_{j=1}^{m} R_j                                      ▷ Set of resources, where m = N · I
Require: K                                                        ▷ Number of tradeoff solutions
Ensure: S = ⋃_{i=1}^{K} sched_W, sched_W = {(A_i, sched(A_i)) | ∀A_i ∈ A}   ▷ Set of K tradeoff schedules
1: function MOHEFT(W,R,K)
2:     Rank ← B-RANK(A)                                           ▷ Order the tasks according to B-rank
3:     for k ← 1, K do                                            ▷ Create K empty workflow schedules
4:         S_k ← ∅
5:     end for
6:     for i ← 1, n do                                            ▷ Iterate over the ranked tasks
7:         S' ← ∅
8:         for j ← 1, m do                                        ▷ Iterate over all resources
9:             for k ← 1, K do                                    ▷ Iterate over all tradeoff schedules
10:                s ← S_k ∪ (Rank_i, R_j)                        ▷ Extend all intermediate schedules
11:                if COUNTRESOURCES(sched_W, m) > N then         ▷ More than N instances used
12:                    T_s ← ∞                                    ▷ Mark schedule as non-valid
13:                    C_s ← ∞
14:                end if
15:                S' ← S' ∪ {s}                                  ▷ Add new mapping to all intermediate schedules
16:            end for
17:        end for
18:        S' ← SORTCROWDDIST(S', K)                              ▷ Sort according to crowding distance
19:        S ← FIRST(S', K)                                       ▷ Choose K schedules with highest crowding distance
20:    end for
21: return S
22: end function
```

we should allow mapping it to resources that provide a tradeoff between the considered objectives.

MOHEFT described in pseudocode in Algorithm 1 extends the original HEFT algorithm by approximating a set of tradeoff solutions K instead of a single one. Similar to HEFT, it ranks first the tasks using the B-rank metric (line 2). However, instead of creating an empty solution as in HEFT, it creates a set S of K empty solutions (lines 3–5). Afterwards, the mapping phase of MOHEFT begins (lines 6–20). MOHEFT iterates first over the list of tasks (line 6) sorted by their computed rank. The idea is to extend every solution in S by mapping the next task to be executed onto all m possible resources and store them in a temporal set S' which is initially empty (line 7). For creating these new solutions, we iterate over the set of resources (line 8) and the solution set S (line 9), and add the new extended intermediate schedules to the new set S' (line 15). This strategy results in an exhaustive search if we do not include any restrictions. Therefore, we save only the best K tradeoffs solutions from the temporary set S' into the set S (lines 18–19). We consider that a solution belongs to the best tradeoff if it is not dominated by any other solution and if it contributes to the diversity of the set. For this last criterion, we employ the crowding distance [10], which gives a measure of the area surrounding a solution where no other tradeoff solution is placed. Our criterion is to prefer solutions with a higher crowding distance, since this means that the set represents a wider area of different tradeoff solutions. The constraint on the number of resources is checked in line 11. If the constraint is not violated, the makespan and cost are computed as before, otherwise they are set to infinite. This will cause the algorithm to discard that solutions later on line 18, producing only tradeoff solutions which use at most N instances. After assigning all the tasks (line 21), the algorithm returns the set of K best tradeoff solutions.

Given a set of n activities and m resources, the computational complexity of HEFT is $O(n \cdot m)$. MOHEFT only introduces two main differences with respect to HEFT: the creation of several solutions in each iteration of the algorithm, and the possibility of considering resources providing a tradeoff solution. These two modifications only require an additional loop in MOHEFT (see Algorithm 1, lines 9 – 16). Considering that the set of tradeoff solutions is K, the extra loop in MOHEFT performs only K iterations, rendering a complexity of $O(n \cdot m \cdot K)$. Usually, the number of tradeoff solutions is a constant much lower than n and m. For example, a workflow can be composed of thousands of tasks and the set of tradeoff solutions can be accurately represented with tens of solutions. Thus, the complexity can be approximated as almost $O(n \cdot m)$, as in HEFT.

IV. Experimental Setup

We describe in this section the experiments carried out for validating the Cloud-aware MOHEFT algorithm.

IV.1 Evaluation Metrics

We consider three criteria for comparing the quality of solutions. First, we consider the shortest makespan of the schedules computed by the three analysed techniques. Second, we focus on the economic aspect of the schedules, analysing the cheapest solution reported by each technique. The idea of these two indicators is to assess the behaviour of the different approaches optimising each individual criterion. Finally, we consider the hypervolume indicator for assessing the quality of computed tradeoff solutions. Second, we analyse the tradeoff solutions for different workflow types. Although we compute the tradeoff between cost and makespan, for the sake of highlighting the potential of the obtained results, we will plot the cost savings versus the makespan deterioration, as percentages relative to the most makespan-efficient solution, computed by HEFT. Third, we study the number and the type of instances selected by the different scheduling solutions computed by the three approaches.

We compare the MOHEFT algorithm with SPEA2* [25], a version of the SPEA2 genetic algorithm proposed in [26] which was shown to outperform NSGA-II and PAES for multi-objective workflow scheduling in [25]. We implemented SPEA2* using the jMetal framework [11], slightly modified to deal with the limitation imposed by commercial Clouds on the maximum number of simultaneous resources. This algorithm requires the same input parameters as MOHEFT and works with a population (set) of candidate solutions which are iteratively recombined with the aim of evolving them towards the optima. In our experiments, we used $K = 10$, apply the recombination operator with a probability of 0.9 and the mutation with 0.5. This configuration is the same one used in the original paper where SPEA2* is described. In order to avoid our conclusions be biased by any hazard effect of this stochastic behaviour, we run SPEA2* for five times and always consider the run producing the front with the largest hypervolume.

IV.2 Workflow Applications

We generated three types of synthetic workflows using the random workflow generator described in [24]. Our interest is to analyse how the number of independent activities influences the scheduling results. Therefore, the defined types are intended to cover a wide spectrum of workflow structures from this point of view:

- *Type-1* where the number of tasks that can be executed in parallel ranges between one and two;

Table 1: Performance and price of various Amazon EC2 instances.

Instance	Mean performance [GFLOPS]	Price [$/h]	GFLOPS/$
`m1.small`	2.0	0.1	19.6
`m1.large`	7.1	0.4	17.9
`m1.xlarge`	11.4	0.8	14.2
`c1.medium`	3.9	0.2	19.6
`c1.xlarge`	50.0	0.8	62.5

- *Type-2* where the number of tasks that can be executed in parallel is high, and the workflow is balanced (same number of tasks in every level);
- *Type-3* where the number of tasks that can be executed in parallel is high, but the workflow is unbalanced (different number of tasks in every level).

We generated the length of every task and the data produced using a Gaussian distribution. For every type, we considered 100 different instances having between 100 and 1000 different tasks.

IV.3 Resource Infrastructure

Amazon EC2 offers fourteen different types of on-demand instances with different performance and price. In [1], Iosup *et al.* evaluated them for scientific computing and reported the average performance in millions of floating point operations per second (GFLOPs) of five different instance types thorough extensive benchmark experimentation. Table 1 summarises the average performance, the price per hour of computation, and the ratio GFLOPs per invested dollar of these resources. In this work, we will evaluate our multi-criteria workflow scheduling method on these five instance types.

We consider that the user has access to the default maximum number of $N = 20$ Amazon instances which can be of any of the five types summarised in Table 1 (i.e. $I = 5$ and $m = N \cdot I = 20 \cdot 5 = 100$). We assume that no public IP addresses are required for running the experiments on the Amazon EC2 infrastructure. Additionally, the output data transfers from Amazon to the outside Internet are constant, take place only at the end of the workflow execution and thus, do not influence the scheduling results. In this situation, we assume in our experiments that the prices for data sent and received are zero: $PI_{R_i} = 0$ and $PO_{R_i} = 0$.

V. Evaluation

We present in this section the evaluation results for the synthetic workflow first, then for the real-world ones. Finally, we analyse how the solutions computed by the algorithms change when the constraint of simultaneously using 20 resources is relaxed.

V.1 *Type-1* Workflows

First, Fig. 1a shows that MOHEFT outperformed SPEA2 in terms of hypervolume for all evaluated *Type-1* workflow instances. We did not include HEFT in this comparison because it only delivers a single solution with the optimal makespan. It is remarkable that, for this workflow type, MOHEFT always computed solutions with the same hypervolume value, meaning that the shape of the optimal set of tradeoff solutions in this case does not vary with the workflow size.

In terms of makespan (see Fig. 1b), all the three methods computed the same solution, which confirms that the performance of MOHEFT does not degrade compared to HEFT. In case of SPEA2*, the results are not surprising since the algorithm is initialised with the solution computed by HEFT. Both MOHEFT and SPEA2 computed the same cheapest schedule illustrated in Fig. 1c, which considers the cheapest instance (`m1.small`) for the entire workflow. This fact is a consequence of the low degree of parallelism of this workflow. The solution computed by HEFT is always the most expensive one.

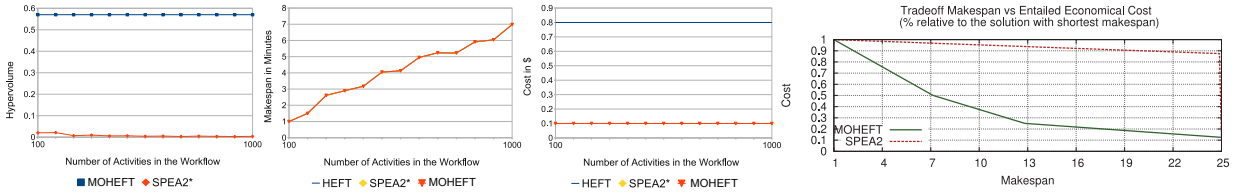
Fig. 1d shows an example of tradeoff solutions computed by MOHEFT and SPEA2*. The higher quality of the solutions computed by MOHEFT can be easily visualised in this chart. In particular, we observe that our method computed a schedule which halves the price of the solution with the optimal makespan by only introducing a 7% of time overhead. In case of SPEA2*, a solution with the same cost would have required an increase of 25% in makespan. These results highlight the importance of the Pareto front as a decision support tool, since computing a single schedule at a time would have hidden this information.

V.2 *Type-2* Workflows

In terms of the quality of the set of tradeoff solutions, MOHEFT has again outperformed SPEA2* for all *Type-2* workflow sizes, as indicated by the hypervolume indicator in Fig. 2a. In this case, different workflow sizes result in Pareto fronts with different hypervolumes, meaning that the shape of the Pareto front for this problem depends on the number of tasks that can be executed in parallel. If we focus on the makespan (see Fig. 2b), it is worth mentioning that MOHEFT and SPEA2* were able in some cases to compute solutions with better makespans than HEFT. The explanation for this behaviour is that, due to its greedy nature, HEFT easily converges towards a local optimum, situation which is overcome by MOHEFT and SPEA2* due to a larger exploration of the search space. In terms of economic cost (see Fig. 2c), MOHEFT and SPEA2* computed the same solution which is a lot cheaper than the solution with the best makespan.

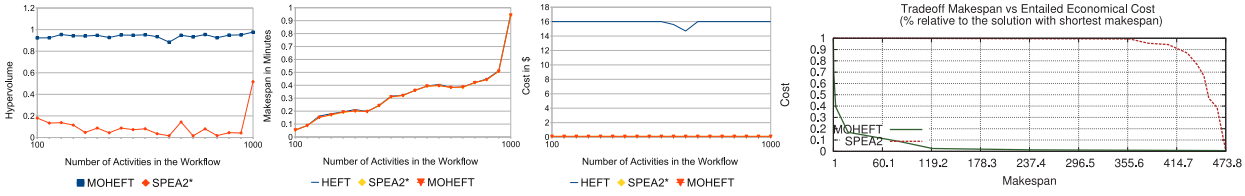
Fig. 2d shows a comparison of the tradeoff solutions computed by MOHEFT and SPEA2*. The differences between both algorithms are even more noticeable than for workflows of *Type-1*. In this case, MOHEFT computed a schedule which reduced the cost by 30% incurring only a 1.4% increase in makespan. Computing a solution of similar price for SPEA2* would have required increasing the makespan by more than 450%. This huge difference between our approach and SPEA2* clearly points out the potential of MOHEFT for multi-objective workflow scheduling in terms of the quality of the computed solutions.

For this workflow type, many solutions computed by SPEA2* required more than 20 resources, thus invalidating its adoption for workflow scheduling in the context of commercial Clouds with limitations on the maximum number of instances that can be simultaneously rented. In particular, 66% of the computed schedules required more than the 20 resource limit imposed by Amazon EC2 (see Fig. 2d). This behaviour does not appear in the solutions computed by MOHEFT or HEFT, which always provided schedules with at most 20 resources.



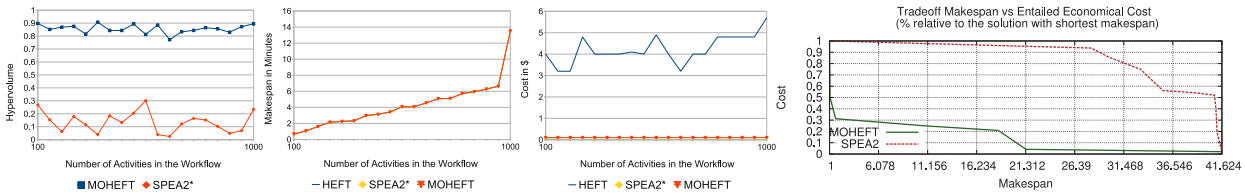
(a) Hypervolume. (b) Makespan. (c) Economic cost. (d) Cost – makespan tradeoff.

Figure 1: Evaluation results for synthetic workflows of Type-1.



(a) Hypervolume. (b) Makespan. (c) Economic cost. (d) Cost – makespan tradeoff.

Figure 2: Evaluation results for synthetic workflows of Type-2.



(a) Hypervolume. (b) Makespan. (c) Economic cost. (d) Cost – makespan tradeoff.

Figure 3: Evaluation results for synthetic workflows of Type-3.

V.3 *Type-3* Workflows

The results for this workflow type, summarised in Fig. 3, confirm the findings of the previous two types.

The hypervolume of the tradeoff sets for the *Type-3* workflows (see Fig. 3a) shows that MOHEFT outperforms SPEA2* also in this case. As in the previous case, the hypervolume reflects a different shape of the Pareto front for workflows with different number of activities. This fact validates the hypothesis that the shape of the tradeoff solutions depends on the number of activities of the workflow that can be executed in parallel. All three techniques computed the schedule which minimises the makespan, confirming again the suitability of this method for workflow scheduling if the user is only interested in optimising this goal. Similar to the previous case, MOHEFT and SPEA2* computed the best solutions in terms of economic cost. The difference between HEFT and the other two methods tends to increase with the number of activities composing the workflow.

An example of the tradeoff solutions computed by MOHEFT and SPEA2* is shown in Fig. 3d. For this workflow type, MOHEFT was able to compute solutions that halve the maximum price with only 1% increase in makespan, while SPEA2* required at least a 40% of extra time for a solution of the same cost. These results indicate once more the better suitability of MOHEFT for multi-objective workflow scheduling on the Amazon EC2 Cloud. In this case, the three techniques always computed schedules meeting the restriction of using at most 20 on-demand instances.

VI. Discussion and Outlook

Designing, optimising, scheduling, and executing scientific applications for heterogeneous computing infrastructures, including production Clouds, involve multiple cycles of code development, small testing followed by real executions, performance monitoring, data collection, optimisation and tuning, which is a cumbersome, tedious, and time-consuming multi-experimental process if not supported by appropriate tools. Working with heterogeneous and dynamic production platforms, such as the European Grid Infrastructure (EGI) or the Amazon EC2 Cloud, brings additional complexity related to performance variability (due to external factors), non-deterministic parallel executions, virtualization overheads, or reliability issues requiring repeated experimentation to produce statistically relevant

results. Repeated experimentation, however, has the drawback of being time and resource consuming (in both manpower and hardware), of disrupting regular production on the platforms, and of limiting freedom in exploration of new cases (e.g. for the sake of curiosity). This issue becomes critical in public Cloud infrastructures, that through their new pay-as-you-go cost resource provisioning model, make the experimentation costs transparent. Finally, energy consumption has recently become another argument against "wasting natural resources" for curiosity science that may yield validated results only in few cases. Faced with this situation, scheduling and executing scientific applications in production distributed computing infrastructures (DCI), including Clouds, has become an increasingly complex multi-objective optimisation problem involving several conflicting metrics for which no appropriate tool support exists due to the increased difficulty in deploying and testing new heuristic methods in real production environments. As a first step in this direction, we proposed a truly multi-objective workflow scheduler called MOHEFT, which extends a well-known list scheduling heuristic in a multi-dimensional objective space of tradeoff solutions. We applied the algorithm in the context of makespan and economic cost optimisation, extended to deal with the realistic constraints imposed by commercial Clouds that restrict the total number of resources that can be simultaneously acquired, but keep their type flexible depending on the temporal needs.

To research and validate such new scheduling heuristics, computer scientists rely nowadays on mathematical models [19, 14], simulators [3, 6, 8], or experimental platforms [5] to reproduce real systems in controlled conditions, which nonetheless remains a challenge [16, 12]. Among these, simulation tools have emerged as important exploration means to facilitate the conduction and management of thousands of experiments, freeing scientists and developers from the complexity and variability of the underlying infrastructure. Simulation tools not only allow easier prototyping and testing of new methods, but also enable their thorough evaluation in situations not easily encountered in real-world scenarios through deterministic and reproducible experiments. Inline with these considerations, we validated and compared MOHEFT with the original HEFT algorithm and with SPEA2*, an extension of the state-of-the-art multi-objective optimisation algorithm SPEA2 by simulating three types of synthetic workflows with different parallelization and work balancing characteristics on Amazon EC2 resources. We showed that the visualisation of the Pareto front can represent a powerful decision making tool for selecting the most appropriate tradeoff solutions. For example, it revealed that certain workflows can be executed twice as cheap by conceding a marginal 5% increase in makespan. In all experiments, MOHEFT computed schedules with the same makespan as the HEFT but with better economic cost, and outperformed SPEA2* in terms of hypervolume used as an indicator of the quality of the set of tradeoff solutions. A visual analysis of the tradeoff solutions revealed that SPEA2* computed in many cases solutions with a higher economic for the same makespan. Finally, our experiments revealed that MOHEFT was able to meet the resource constraints imposed by current commercial Clouds, while SPEA2* failed on this issue.

Our validation, however, is limited to simulation of synthetic workflows and lacks a real-world validation. A reason for this is a major drawback of the existing simulation tools is that they are not properly tuned for production platforms. As argued in [7], most of the existing works validate their research (including scheduling) based on simulation without giving sufficient proofs that the simulator accurately reproduces the behaviour of a real infrastructure, which makes the accuracy and relevance of the results doubtful. Furthermore, most works do not validate the simulated results by comparing them with real executions in a real infrastructure. A reason that brought to this unfortunate situation is the lack of integrated tools that close the cycle between experimentation of new basic research methods (such as investigation of new scheduling optimisation algorithms), their extensive and accurate validation through realistic infrastructure modelling and simulation tools, and finally their integration and deployment on the real platform for production runs delivering the expected improved performance.

We plan in the future to validate our proposed method for real-world scientific workflows on production Cloud infrastructures, which is a real challenge due to the lack of appropriate validation tools. Faced with this research problem, University of Innsbruck together two INRIA divisions (CREATIS – CNRS and IN2P3 Computing Center) aim to research a framework enabling lightweight exploration and evaluation of new scheduling heuristic methods for scientific applications on real Cloud infrastructures through extensive realistic simulations, before deploying them for production runs as illustrated in Figure 4. This work therefore aims to reduce the experimentation costs and contribute to shortening the application lifecycle from its design to production operation, maintenance and tuning. We intend to research methods to build *realistic simulations of real DCIs* from observations and existing simulation toolboxes, with particular focus on the EGI Cloud platform. We will specifically focus on the simulation of the *Virtual Imaging Platform (VIP)* [15], one of the most used scientific computing platforms on the EGI that facilitates sharing of medical image simulators and digital models of the human body. We intend to extend our objective space with other metrics of interest alongside makespan and economic cost, such as energy consumption, reliability, utilisation, fairness, security, and any other Quality of Service or functional application-specific parameter.

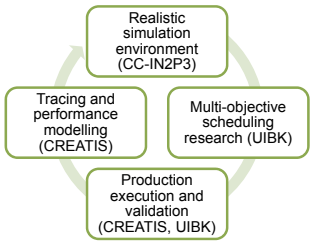


Figure 4: Closing the modelling and simulation – research – production execution cycle.

Acknowledgements

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

References

[1] I. Alexandru, S. Ostermann, M.N. Yigitbasi, R. Prodan, T. Fahringer, and D.H.J Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions onf Parallel and Distributed Systems*, pages 1–16, 2010.

[2] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks, DSN'04*, Firenze, Italy, June 2003. IEEE.

[3] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. Optorsim – a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.

[4] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and c onceptual comparison. *ACM Computing Surveys*, 35(3):268 – 308, 2003.

[5] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[6] Rajkumar Buyya and Manzur M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[7] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and systems. *Journal of Parallel and Distributed Computing*. in press.

[8] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a generic framework for large-scale distributed experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.

[9] Carlos A Coello Coello, Gary B Lamont, and David A Van Veldhuisen. *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[11] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[12] Dick H.J. Epema. Twenty years of grid scheduling research and beyond (keynote talk). In *CCGrid'2011*, pages xxxi – xxxiii, Ottawa, CA, may 2012.

[13] Saurabh Kumar Garg, Rajkumar Buyya, and H. J. Siegel. Scheduling parallel applications on utility grids: time and cost trade-off management. In *Proceedings of the Thirty-Second Australasian Conference on Computer Science - Volume 91*, ACSC '09, pages 151–160, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.

[14] T. Glatard and S. Camarasu-Pop. Modelling pilot-job applications on production grids. In *7th international workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (Heteropar 09)*, Delft, The Netherlands, 2009.

[15] T. Glatard, C. Lartizien, Bernard Gibaud, R. Ferreira da Silva, G. Forestier, F. Cervenansky, M. Alessandrini, H. Benoit-Cattin, O. Bernard, S. Camarasu-Pop, N. Cerezo, P. Clarysse, Alban Gaignard, Patrick Hugonnard, H. Liebgott, S. Marache, A. Marion, J. Montagnat, J. Tabary, and D. Friboulet. A virtual imaging platform for multi-modality medical image simulation. *IEEE Transactions on Medical Imaging*, 32(1):110–118, 2013.

[16] Jens Gustedt, Emmanuel Jeannot, and Martin Quinson. Experimental validation in large-scale systems: a survey of methodologies. *Parallel Processing Letters*, 19(3):399–418, 2009.

[17] Mourad Hakem and Franck Butelle. Reliability and scheduling on systems subject to failures. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, pages 38–, Washington, DC, USA, 2007. IEEE Computer Society.

[18] E. Ilavarsan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3(2):94–103, 2007.

[19] J.T. Moscicki, M. Lamanna, M. Bubak, and P.M.A. Sloot. Processing moldable tasks on the grid: Late job binding with lightweight user-level overlay. *Future Generation Computer Systems*, 27(6):725–736, 2011.

[20] Munindar P. Singh and Mladen A. Vouk. Scientific Workflows: Scientific Computing Meets Transactional Workflows, 1996.

[21] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[22] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260 –274, mar 2002.

[23] J.D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[24] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. In F. Xhafa and A. Abraham, editors, *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 109–153. Springer Berlin, 2008.

[25] Jia Yu, Michael Kirley, and Rajkumar Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 10–17, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.